The IBM Victor V256 partitionable multiprocessor

by D. G. Shea
W. W. Wilcke
R. C. Booth
D. H. Brown
Z. D. Christidis
M. E. Giampapa
G. B. Irwin
T. T. Murakami
V. K. Naik
F. T. Tong
P. R. Varker
D. J. Zukowski

Victor V256 is a partitionable message-passing multiprocessor with 256 processors, designed and in use at the IBM Thomas J. Watson Research Center. Our goals are to explore computer architectures based on the message-passing model and to use these architectures to solve *real* applications. We present the architecture of the Victor system, particularly its partitioning and nonintrusive monitoring. We discuss some of the programming environments on Victor, such as E-kernel, an embedding kernel developed for the support of program mapping and network reconfiguration. We review applications developed and run on Victor and discuss a few in depth, concluding with insights we have gained from this project.

1. Introduction

• *Victor*

The Victor project at the IBM Thomas J. Watson Research Center started with the dual objectives of investigating highly parallel, message-passing, distributed-memory MIMD (multiple-instruction/multiple-data) machines and using these machines to solve real problems. Our previous experience with the design of uniprocessors led us to an understanding that the data access latency and bandwidth between memory and processor is of supreme consequence in determining overall system performance [1]. Therefore, we decided that to obtain the significant speedups that are expected for highly parallel systems, an architecture based on distributed memory (hence, the message-passing communication model) was important.

In late 1986, we embarked on the design and rapid construction of a message-passing multiprocessor with off-the-shelf components. An analysis of the available single-chip processors resulted in our choice of the Inmos™ transputer [2], a specialized microprocessor with interprocessor-communication support incorporated on the chip. By the summer of 1987, we had produced an operational 32-processor system called Victor V32. This system provided us the ability to explore message-passing and to run real code. Fractals, ray tracing, and Monte Carlo nuclear physics, described in the applications section, were the first programs to run on V32. In addition, V32 was used for the initial development of E-kernel, which is described in the system software section.

The original Victor V32 prototype led to Victor V256 (with 256 processors) and an associated family of smaller systems based on the same architecture and card set. Early motivations and a description of the project can be

found in [3]. The goals for the Victor project were to better understand what classes of problems are suited to this type of architecture and to provide an early "platform" for application development for massively parallel systems.

This paper discusses the Victor hardware architecture, in particular the capabilities for nonintrusive monitoring and partitioning, system software components, and some applications that have been created and run successfully. The applications have spanned the communication spectrum from waveform-relaxation circuit simulation and logic-fault simulation, which have highly irregular communication patterns, to Monte Carlo simulations of nuclear physics problems, which have extremely low communication requirements and are consequently limited in performance by only the processing power available at each node.

- *Related work*

Before describing in detail the Victor system, we briefly review some related systems work. The pioneering machine prototype credited with generating much interest in distributed-memory machines was the Cosmic Cube at the California Institute of Technology, a 64-node machine based on the Intel 8086 processor and interconnected as a hypercube [4]. Seitz stated in 1985 that the Cosmic Cube nodes were designed as a hardware simulation of what we could expect to be integrated into one or two chips in about five years [4]. The existence of the Inmos transputer has substantiated this claim [2, 5]. Another chip that has appeared to support Seitz's statement is the iWarp processor, which integrates high-speed computation and high-speed communication capability in a single component, achieving 20 MFLOPS and 20 MIPS per node [6]. The iWarp processor is the basis for the iWarp distributed-memory architecture developed jointly by Carnegie Mellon and Intel Corporation. There continues to be much active research in the development of message-passing architectures. Dally from MIT is working on the design of the J-Machine, a fine-grained concurrent multiprocessor that provides low-overhead primitive mechanisms for communication, synchronization, and translation [7]. We now briefly describe three efforts that, in various ways, are similar to the Victor approach—Armstrong, Hathi-2, and Esprit.

Armstrong

The primary goal of the Armstrong project was to construct a useful, controllable, research-oriented parallel processing test bed [8]. The system, developed in the Laboratory for Engineering and Man/Machine Systems at Brown University, has on the order of 100 nodes. Each Armstrong node contains a Motorola 68010 CPU, one-half megabyte (MB) of RAM, and eight I/O links. The interconnection is point-to-point and can be manually reconfigured using "patch-cords." Floating-point performance is achieved through the use of the National 32081 coprocessor.

One goal of the prototype was to allow the same application code to be run without modification on any network configuration. The Armstrong operating system supports the reconfigurable network by providing location-independent communication. From the programmer's point of view, interaction between processes is the same, whether or not they are on the same node. The operating system determines the route by using the shortest path available. It determines the shortest path between nodes dynamically by broadcasting routing packets periodically; on Armstrong, this is done every 20 seconds. This allows the system to adapt to changes in topology dynamically.

A fundamental difference in the approaches of the Armstrong operating system and E-kernel (which we discuss in the system software section) is that the task graph (which represents the communication structure) of an Armstrong application program is essentially viewed as a complete graph. No information from the application lets the operating system take advantage of the topology of the application task graph. With E-kernel, the user writes the code according to the natural task graph of the application, and E-kernel presents this information to the system in order to allow an efficient mapping of the program onto the system.

Hathi-2

Hathi-2 is a reconfigurable, general-purpose, multiprocessor system that was developed in the Department of Computer Science at Abo Akademi and the Technical Research Centre of Finland in Oulu [9].

The current implementation uses 100 transputer nodes, each with 256 kilobytes (KB) of memory. Several users can execute applications on Hathi-2 simultaneously. Hathi boards are interconnected to form a 4×6 torus. The system may be divided into partitions, each with one or more boards. One particular board out of the 25 is not part of the torus connection; its links are not connected to any other board, and it always forms a separate partition consisting of four nodes. Therefore, the Hathi system always consists of at least two partitions, and the largest single partition that can be formed consists of 96 transputers. Partitioning is done with a set of manual switches on the backplane of Hathi-2 and cannot be changed during run time, because the system must be rebooted after partitioning. In comparison, the granularity of partitioning on Victor is at the node level, so the partition size, which is under software control, can vary from one node to the maximum system size of 256 for V256.

The Hathi approach in development was similar to ours for Victor; we quickly developed the prototype of a 32-node system (V32), then designed V256. They built a Hathi-1 system, which was a 16-node system using off-the-shelf Inmos B003 boards. Hathi-2 is logically divided into two separate subsystems: the multiprocessor system and the control system. The former contains 100 T800 transputers, and the latter consists of 25 T212 transputers, C004 crossbar switches, monitor hardware, and the interrupt system. The T212 transputers in the control system are connected to form a ring and are controlled by a separate host computer. The control system has two functions:

- *Reconfiguration:* Controls the setting of the C004 switches.
- *Monitoring:* Forwards monitoring data from the multiprocessor system to the control-system host.

For monitoring, each T800 transputer writes into its set of FIFO registers, which are read by the T212s, and data are sent on the control ring to the host. Information is collected about the utilization of the CPU and the communication links, from the processor "load meter" and FIFO buffer. The processor load is determined by dedicated hardware that detects activity on the memory bus; this is achieved by dedicated hardware and is nonintrusive to the application. This part of the monitoring logic is very similar in approach to the Victor monitor. All additional monitoring information in Hathi, such as link activity, however, is intrusive. This causes overhead, since the nodes need to collect the information and write it to their FIFO registers. This differs from Victor, in which all monitoring is nonintrusive.

Esprit

Esprit is the European Strategic Program for Research and Development in Information Technology. The Esprit P1085 program had the objective of developing a MIMD multiprocessor machine with supporting software, to demonstrate that high performance can be achieved over a wide range of applications [10]. The result was a modular, hierarchical architecture based on reconfigurable nodes of transputers. In this system a node consists of 16 worker transputers, each with 256 KB of memory. One additional node in the system has 16 MB of memory, which is used for storing and distributing data. Each worker transputer has four links, with each link connected to a 72 × 72 VLSI switch controlled by another transputer. Southampton University designed the basic reconfigurable "SuperNode" cluster architecture [11]. Additional internode switches are used to implement a three-stage Clos network [12] for reconfiguring nodes. This flexibility allows one to construct any network that can be built with nodes having four network connections. The system provides more reconfiguration capability in hardware than Victor but lacks the hardware monitoring facilities.

2. Victor system

Victor is a family of experimental, partitionable, distributed-memory, message-passing multiprocessors. Victor V256, the largest in the family, is a 256-processor system containing one GB (gigabyte) of main memory and 10 GB of distributed DASD. Other members of the Victor family include a (64-processor) Victor V64 system and several identical (16-processor) Victor V16 systems. This section addresses the system hardware design and the system software developed and run on Victor.

- *System hardware*

Here we present a view of the hardware, from the high-level system architecture to the individual node architectures. There are four distinct types of nodes in the Victor system: *processor*, *disk*, *host*, and *graphics*, all using the Inmos T800 transputer [2]. With these as building blocks, a wide range of system capabilities can be provided. We discuss in detail the processor and disk node architectures, highlighting the partitioning and monitoring capabilities of Victor. We conclude the hardware section with a short discussion of implementation details.

System architecture

The processor nodes of V256 form a 16 × 16, two-dimensional mesh. **Figure 1**, which is a photograph of the Victor monitor, presents a snapshot of system activity. Each box in the central area represents a processor node. Each white bar represents an active communication link. The disk nodes are represented in the figure by the row of 16 white, rounded-corner boxes at the top of the screen. They form a 17th row, with connections to the upper and lower boundaries of the mesh, thus closing the mesh into a cylinder. At the time represented by Figure 1, no request is being made to the file system; therefore, no active links are shown between the processor and disk nodes. The five host nodes are shown near the corners of Figure 1 and at the left center. Four graphics nodes are connected to the left and right sides of the mesh.

V256 is a partitionable multiuser system for up to four concurrent users and the superuser (a fifth user). A user is assigned a host node and a graphics node, and selects an arbitrarily shaped contiguous region of the processor mesh. In Figure 1, four individual users are assigned hosts H1 through H4, each owning a region of the system represented by a group of boxes of a different color. (Although the shape of partitions may be arbitrary, most applications that have been run on Victor have used rectangular partitions. The disk nodes are owned by the superuser, at host H0, and act as servers, handling all file

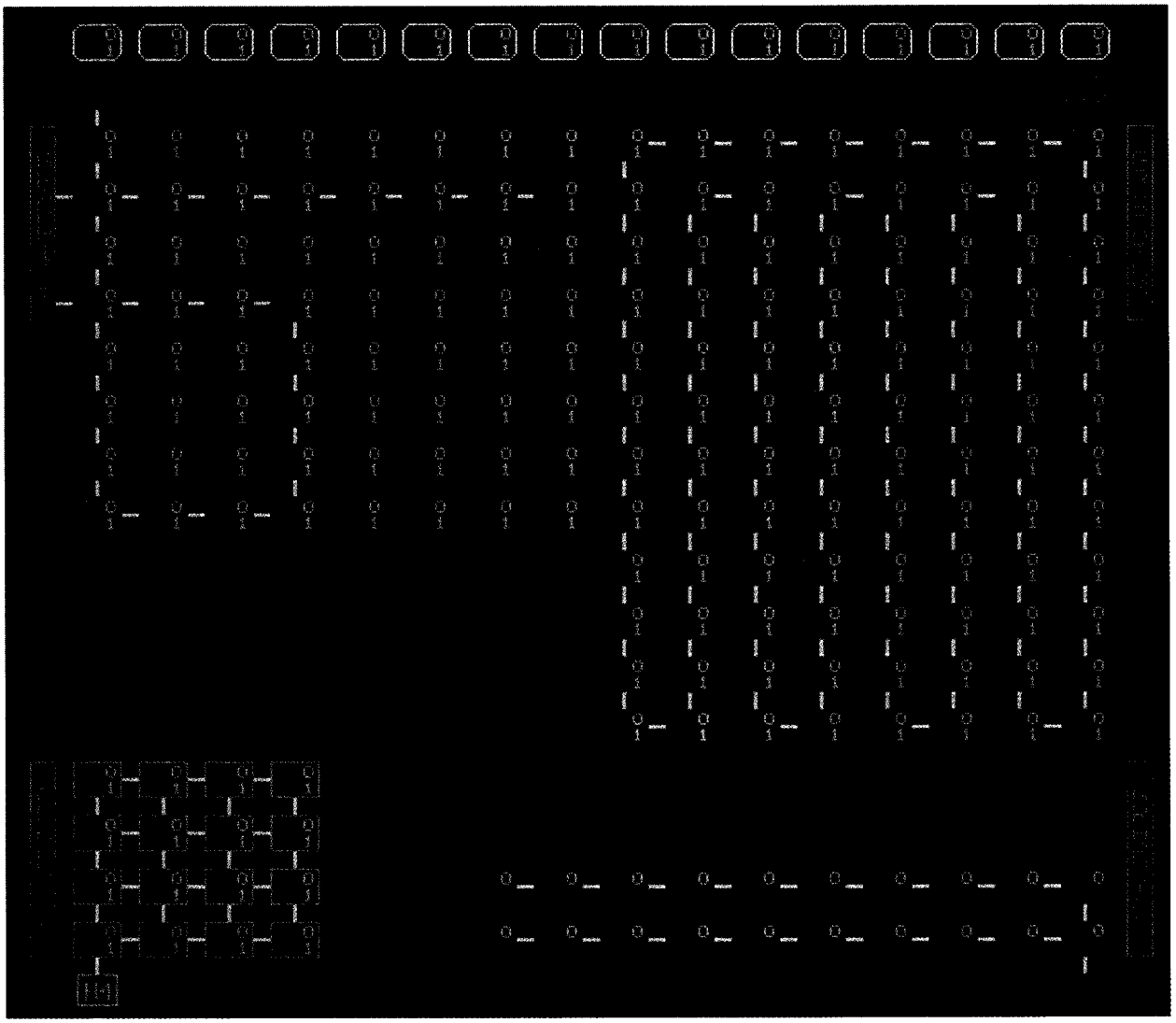

Figure 1

Photograph of Victor V256 monitor.

system requests from the users. In the configuration of Figure 1, each of four regions was active with a different program. Hardware discussed below provides this partitioning and prevents users from interfering with one another. The mesh topology, coupled with the partitioning strategy, furnishes a user the same general network characteristics at any one of the four partitions. On the basis of our experience with V32, we decided that the practical advantages of a highly regular topology outweighed the benefits gained from using a more complex topology of smaller diameter (the maximum distance any message may have to travel).

Node architecture

Figure 2 is a block diagram of a Victor processor node. Each node is based on a 20-MHz T800 transputer chip, which includes a 64-bit floating-point unit and four transputer links. The memory subsystem provides 4 MB of memory per node. Memory access times are given in **Table 1**. A standard 32/39-bit error-correcting-code (ECC) circuit increases system reliability, a very important consideration in a system with 1 GB of dynamic RAM running for weeks at a time on a single problem.

Each processor node contains a memory-mapped 16-bit event status register (*EvSR*) that contains information

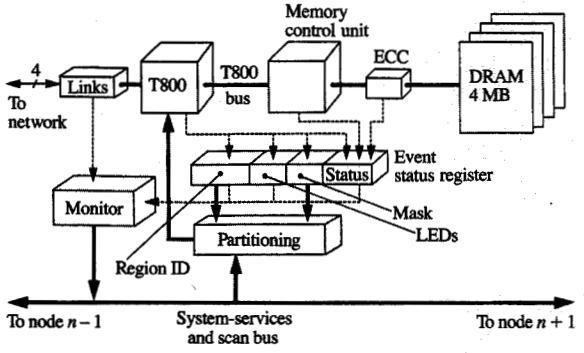

Figure 2

Victor V256 processor node architecture.

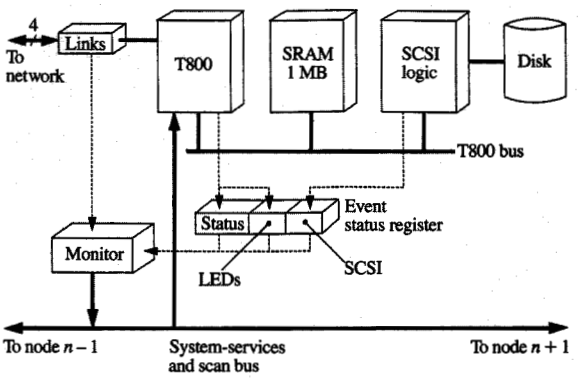

Figure 3

Victor V256 disk node architecture.

about memory errors and transputer errors, various mask bits, two software-programmable flags connected to LEDs (also displayed as *0* and *1* in Figure 1), and a bit indicating whether this node is currently being polled by the monitor. In addition, this register has a four-bit *region ID* field (one bit per host) that indicates the present owner of this node.

It was desired to have sufficient disk I/O in V256. Past experience with I/O traffic in scientific code suggests that a system with the performance of V256 should ideally have up to 100 MB/s in I/O capability [13]. The capacity of the 16 medium-performance disks integrated into V256 falls short of this ideal. As discussed in the subsection on the file system, even though the transfer rate from disk drive to disk node memory was of the order of 1.0 MB/s, the effective transfer rate from disk node to processor node was approximately 310 KB/s. One application to use the file system is waveform relaxation. The combined data rate of the 16 disk nodes met the requirements for that application but is not sufficient for I/O-intensive applications. We hope to learn how to support parallel I/O by using the disk resources currently available on V256. For instance, if one had the resources to add 64 disk nodes to V256, where should they be placed?

Figure 3 is the block diagram of a disk node, which is also based on the T800 transputer. The disk-node memory consists of 1 MB of static RAM. A SCSI bus controller is memory-mapped onto the address space of the T800; it uses asynchronous direct memory access data transfer at a maximum rate of 1.0 MB/s. The disks used in V256, V64, and all V16 systems are IBM 600MB and 300MB drives.

Table 1 Victor V256 processor node memory access times.

Memory operation	*Time* (ns)
Read	300
Write word	250
Write byte	350

Partitioning and monitoring

In V32, we used a simple switch to allow multiple hosts to be attached to the system. Although this made it easy to switch hosts, only one host could use V32 at any one time. For this reason and because of work in partitioning by Ma and Krishnamurti [14], a goal for V256 was to make the system partitionable. Logic on each processor card allows each node on that card to be *claimed* by any of the host nodes. Once claimed, a processor blocks requests from other hosts, but the superuser on host H0 can always gain control of the entire system.

A host can acquire a partition of arbitrary size, from one node up to all 256 nodes, the only restriction being that there must be a connected path of links from every node in the partition back to the host. Once the user of a given partition is done with some or all of its nodes, they are released back into the pool of nodes available to be claimed by another host.

In V32, several buffered LEDs per node were hard-wired to various signals, giving some visual indication of the activity on the node. Though primitive, this provided users with a surprising amount of information about the

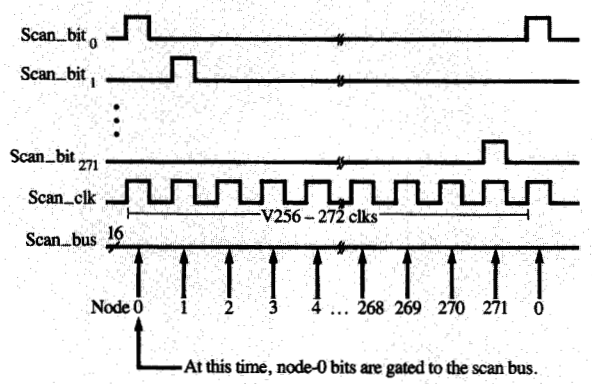

Figure 4

Timing diagram for Victor V256 monitor scan bus.

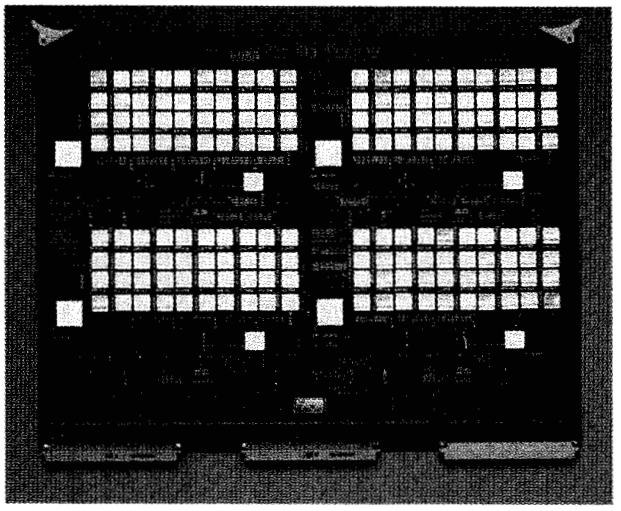

Figure 5

Victor V32 four-processor card.

progress of their programs. For V256, the hardware monitoring was extended and is implemented via a data-collection bus (the *scan bus*) that is completely independent of the regular transputer communication links and is controlled by a dedicated IBM PS/2® monitor processor. All nodes (256 processor nodes and 16 disk nodes) are connected to this systemwide scan bus, which carries 16 bits of data from each of the nodes. These include a subset of the event-status bits and bits conveying such performance information as external-memory usage and link activity. The dedicated PS/2 monitor generates two signals to control the monitoring: Scan_bit and Scan_clk (as shown in **Figure 4**). The Scan_bit is "passed" from node to node at a rate determined by Scan_clk. When a node is "in possession" of the Scan_bit, it gates its status bits onto the scan bus. As shown in Figure 4, a Scan_bit is injected to processor 0 every 272 scan-clock cycles. The monitor processor continuously displays the received status information for the entire system graphically and optionally files it for later analysis.

System implementation

The logic for the Victor machines is packaged on circuit cards such as that shown in **Figure 5**, which shows a processor node card from the original Victor V32 machine. The card contains four independent processor nodes, with each node occupying about 4 × 6 inches. We used the same card size and node layout for the V256 processor card, and it too contains four processor nodes. The disk node uses the same size card, but each card contains only one disk node.

The host nodes and graphics nodes for V256 were purchased from vendors of transputer-based hardware. A small number of modifications were necessary to make these usable in our system—in particular to make them conform electrically to our differentially driven communication-link implementation.

The Victor V256 system is packaged in four 19-inch racks, as seen in **Figure 6**. Two of the racks accommodate the 16 disk-node logic cards and their disk drives, while the other two racks house the 256 processor nodes and power supplies. The packaging is designed for a maximum power dissipation of 20 kW and is air-cooled. It occupies a small machine room, with the hosts located in an adjacent room. Remote users gain access through local-area networks.

Several special measures were taken to enhance system reliability so that V256 could solve complex problems with very long run times. These measures include the following:

- Standard 32/39 ECC code provides single-bit correction and double-bit detection of memory errors.
- All intercard signals (with the exception of those for the monitor scan bus) are repowered with TTL-level differential drivers/receivers.
- A digital filter is included on the system reset lines. This filter is designed to remove noise pulses that would otherwise cause system failures, with no possible recovery.
- All cards run asynchronously. Although the four processor nodes that are packaged on a card share a common oscillator, a circuit is provided on each card to phase-shift the clock before it drives each node. This phase shift guarantees that the nodes will refresh their

DRAM memories at different times and reduce the peak current demand of the card during refresh.
- Thermal design is conservative. The hottest air temperature within the system ranges from 45 to 50°C.

These features, plus engineering attention to logic and card design details, have produced a family of Victor systems with excellent proven reliability characteristics. Since May 1989, when the 256-processor-node system became operational, we have seen an extremely small number of memory errors and only one link failure, all of which were fixed in minutes by plugging in a replacement card.

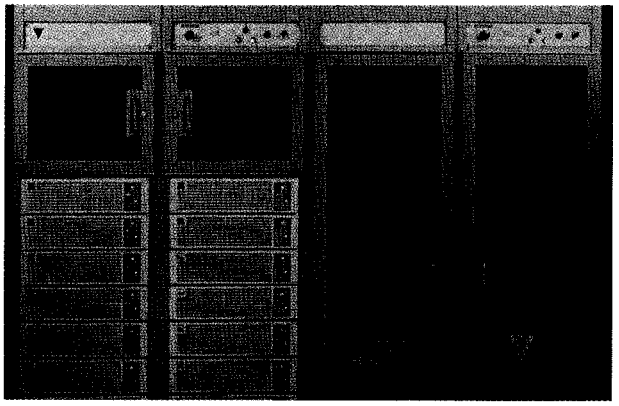

Figure 6

Victor V256 system.

- *System software*

Different message-passing programming environments have been used on Victor to support the applications discussed in Section 3. We now discuss a few of them. The communicating sequential processes model, which influenced the architecture of the transputer, is discussed first. We then consider E-kernel, an embedding kernel developed in the Occam language for experimenting with program mapping and network reconfiguration on Victor. Next we discuss the Express environment, which had its roots in a university project and has been ported to numerous distributed-memory machines, including Victor. We conclude this section with a brief discussion of the Victor file system.

Communicating sequential processes

As we stated in the Introduction, the Inmos transputer was the choice of processor for Victor V32 in late 1986. At that time, the most developed programming language for the transputer was Occam [15, 16], which we chose for our early applications. The language is built on the communicating sequential processes (CSP) paradigm from the work of C. A. R. Hoare [17, 18]. Fundamental principles of this paradigm are that input and output are basic primitives of programming and that parallel composition of communicating sequential processes is a fundamental structuring method. Occam is a high-level language based on the concepts of concurrency and communication, which enables the behavior of concurrent systems to be explicitly programmed and controlled. In Occam, communication between concurrent processes is implemented using *channels*. Each channel provides a one-way, unbuffered connection between two concurrent processes.

A process performs a sequence of actions and then terminates. Each action may be an assignment, an input, or an output. An assignment changes the value of a variable; an input receives a value from another process on a channel; and an output sends a value on a channel. The channel is the synchronization device for coordinating concurrent processes. Since there is no buffering, a channel behaves as a read-only device to a receiving process and as a write-only device to a transmitting process. When both the input process and the output process are ready to communicate on the same channel, the value from the output process is sent to the input process via the channel. When this action is complete, both processes are ready to continue.

A process can be a single primitive process statement, a group of statements, or a group of processes. Functionally, a process is a group of statements that share the same context. Processes are connected to form concurrent systems; the control flow of a process is coordinated by *constructors*, which are used to combine processes to form larger processes. Some example constructors are SEQ (which executes its component processes one after another) and PAR (which causes its component processes to be executed concurrently). We have used the Occam language to develop an embedding kernel that is now discussed.

E-kernel

The better the match between the communication requirements of parallel programs and the communication facilities of a parallel system, the better the potential performance of that system. To achieve a good match on Victor, we designed and implemented E-kernel, an embedding kernel, intended to relieve the user of the concern of optimizing program communication for the system network topology at hand and to allow the user the freedom of using a more natural communication topology for the particular application code being developed. The communication structures provided by E-kernel to the application program are meshes and toruses, representing

many of the intrinsic communication patterns (task graphs) found in scientific applications [19].

E-kernel is used both to optimize the performance of an application running on a Victor partition and to experiment with program performance for systems with different network topologies. E-kernel has two phases. The first, *program mapping*, embeds (maps) the application task graph onto a chosen system network topology, which may be a 2D mesh, a ring, or a linear array. E-kernel assumes that the program task graph has the same number of nodes as the number of processors in the chosen network and tries to place the communicating processes of the application as close together as possible in the system network in order to minimize the maximum distance between any two communicating processes. In its second phase, *network reconfiguration*, E-kernel embeds the chosen network topology onto the 2D mesh of a partition on Victor. The benefit of providing software support for network reconfiguration is to show the effects of different system network topologies on performance. Through the use of E-kernel, without any alteration to the application code, the same program runs on Victor regardless of whether the chosen system network is a 2D mesh, a ring, or a linear array, while the communication optimization of the program is automatically attempted by E-kernel on the different network topologies.

The degree of success in the attempt of E-kernel to optimize the communication depends largely on its embedding functions, which determine the distances between the communicating processes, and its routing strategies, which determine the amount of contention. Further details are provided in [19–22]. E-kernel was developed in Occam and allows the application program to be written in Occam, C, FORTRAN, or Pascal.

Express

A large body of work involving distributed-memory machines, particularly hypercubes, was performed in the 1980s at the California Institute of Technology and is described in [23]. Fox et al. have developed a concurrent programming environment referred to as CrOS III (hypercube crystalline operating systems). In CrOS III, they grouped commonly used communication sequences into "conceptual units." They believe that collectively these communication routines have the advantage of guiding a programmer's thinking toward concurrency while casting the communication operations in a form that may be efficiently implemented on actual machines.

An area of interest they have pursued is that of *loosely synchronous* communication. This refers to the class of problems in which some parameter can be used to synchronize the different processes of the decomposed computation. This computational parameter corresponds, for example, to time in a physical simulation or iteration count in the solution of systems of equations by relaxation.

This work of Fox et al. has continued at Parasoft, resulting in the Express environment, details of which can be found in [23, 24]. One programming model that Express provides is CUBIX, an operating system server that allows distributed applications full access to the operating system resources available on the host computer. Express also provides performance analysis by means of profiling utilities that monitor execution, communication, and events. Several applications running on Victor are programmed in the Express environment.

File system

The hierarchical file system has been developed to provide access to the 10 GB of disk storage available on Victor V256. The file system is a shared resource and is accessible to any region that has a path to the *disk subsystem*, described in the earlier subsection on system hardware. The file system runs independently on each of the disk nodes in the system and handles requests from any application program for data stored on that particular disk node. This is a step toward a truly distributed file system which would run across all disk nodes and manage file storage between disks as well as on individual disks.

We see a usable transfer rate between the file system executing on a disk node and an application running on a processor node of about 310 KB/s. Therefore, the maximum aggregate bandwidth from the file system to an application is approximately 5 MB/s if all 16 disk nodes can be utilized effectively. This number reflects the overhead associated with the file system software and the application program calls to this software. It also reflects degradation due to buffering and routing the data through the message-passing network.

3. Applications

Applications developed for Victor range in scope and intent from exercising the prototype system to efficiently solving complex real-world problems. Initially, development effort focused on applications that are relatively easy to parallelize in order to gain an empirical understanding of problem decomposition, distributing computation across nodes, and balancing communication between nodes. This experience eventually led to the design and implementation of parallel programs on Victor V256 that compare favorably in execution time and capacity to similar sequential code on state-of-the-art mainframes.

In the following, we describe in some detail our experience in implementing applications on Victor. The eleven applications described here represent a wide cross section of engineering and scientific disciplines. In each case, salient points of the problem are described, followed

by a discussion of some of the implementation aspects and the performance results. The total number of processors used in each application varied, depending on the machine sizes available at the time of implementation as well as on the peculiarities of the application. In most cases, the parallel performance is described in terms of the speedups achieved. The speedup using P processors is defined as the ratio of the elapsed time for computing the problem on one processor to the elapsed time for computing the same problem using all P processors. In cases where the problem could not be solved on one processor because of memory limitations, a smaller problem was solved on one processor, and the times for the larger problem were extrapolated.

- *Fractals and ray tracing*

The first applications written for Victor involved problems that could be divided into independent "parcels" of computations. Among these were the implementation of fractals and ray-tracing algorithms. Here, complete world models were replicated in each processor. One of the processors was designated as the master node from which the remaining processors obtained new parcels of computations whenever they were free. In these applications, the time to evaluate the parcels varies widely; hence, as far as parallelism is concerned, the main issue is how to achieve workload balance. A simple dynamic scheme was found effective, in which the total workload was subdivided into many more parcels than there were processors. When a processor finished the computations associated with a parcel, it requested an additional parcel from the master node. All the communication overhead consisted of acquiring the parcels. On the whole, the overhead was negligible compared to the computations associated with the parcels. As a result, close to linear speedups were obtained. The programs were written both in Occam and in Pascal.

- *Monte Carlo nuclear physics*

Another interesting early application was a Monte Carlo solution of a nuclear physics problem. The code used was a very simplified version of one of eight nuclear physics programs that had been studied to determine their suitability for running on message-passing multiprocessor systems [25]. Using Monte Carlo techniques, this code calculated the evaporation of neutrons and light charged particles from nuclear heavy-ion reaction products. The code operated in three distinct phases: initialization, computation, and tabulation of final results. In all nontrivial cases, the communication-free computation phase completely dominated the run time, thus leading to nearly linear speedups. Other than workload balancing, the only nontrivial issue in parallelizing the serial code was the well-known problem of concurrent creation of pseudorandom numbers, which was solved using staggered starts [23]. The original serial FORTRAN code was rewritten in Pascal.

- *Neural network simulation*

The simulation of neural network models on Victor was the first complete application in which the ratio of communication to computation was significant. This program involved the application of the well-known back-propagation algorithm to the noisy-character-recognition problem [26]. The solution proceeded in two phases. In the first phase, the parameter space (learning rate and momentum) of the back-propagation algorithm was explored on multiple processors of Victor. The entire neural network was represented on each processor. At the end of the first phase, parameters were determined that were used in the second phase. In the second phase, the algorithm was recast, distributing the neural network over the two-dimensional Victor mesh. Each neural-network layer was mapped onto a row of the mesh. A centralized control structure was chosen for the program; as a result, all message-passing took place via the host processor, making it a bottleneck and limiting speedups to about 16 on a 32-processor system. The code was written in Occam and C.

- *Computational fluid dynamics*

The focus of this work was to study the performance issues involved in implementation of the implicit-scheme-based computational fluid dynamics (CFD) applications on message-passing systems. The ARC-3D program developed at NASA Ames Research Center was used as representative application code. Using implicit numerical methods, this code solves the three-dimensional Euler and Navier–Stokes equations for compressible flow of gas over a solid body. See [27] for details on the numerical methods used in ARC-3D and [28] for a discussion of the parallelization issues. Detailed timing measurements were carried out, for both the execution of the application as a whole and the four major phases of computation. Analyses were performed for two types of partitioning schemes and the choice of algorithms for solving the implicit systems. The salient points of this work are described below.

Partitioning schemes Two classes of partitioning schemes were considered: *unipartitioning* and *multipartitioning*. In the former case, the entire computational domain is divided into P partitions, where P is the number of processors, and each processor is assigned one partition. Under the multipartitioning scheme, the domain is divided into some multiple of P partitions, and each processor is assigned more than one such partition. Three different forms of unipartitioning schemes (1D, 2D, and 3D) and two types of multipartitioning

Table 2 Speedups and percentage of time spent on each task for a computational fluid dynamics program with 30 × 12 × 30 grid.

Task	*No. of PEs*	*Speedup*		*Percentage of total time*	
		2D unipartitioning	*2D multipartitioning*	*2D unipartitioning*	*2D multipartitioning*
BC	4	3.28	2.77	1.20	1.31
	16	10.34	2.55	1.34	3.37
RHS	4	3.70	3.45	36.70	36.24
	16	13.54	8.17	35.38	36.27
Implicit	4	3.79	3.48	59.78	60.02
	16	13.39	9.73	59.70	50.79
Update	4	0.90	0.79	2.31	2.43
	16	2.05	0.47	3.58	9.56
Complete time step	4	3.69	3.40	100.0	100.0
	16	12.99	8.04	100.0	100.0

schemes (2D diagonal and 3D diagonal) were considered. In the 2D multipartition case, the computational domain was divided into P^2 partitions and each processor was assigned P such partitions. In the 3D multipartition case, the computational domain was divided into $P^{3/2}$ partitions and each processor was assigned $P^{1/2}$ such partitions. In general, unipartitions have smaller communication overhead, whereas the multipartitions have minimum data-dependency delay effects—the delays that are inherent in implicit computations. See [28] for a detailed discussion on the trade-offs involved. Both the 3D partitioning schemes are tolerant to load imbalance. See [29] for the details on the performance effects of load imbalance on computation and communication overheads. Depending on the choice of algorithms, each type of partitioning has certain memory requirements. Overall, the 2D and 3D type of unipartitioning and the 3D multipartitioning "scale" well both in terms of the execution time and memory requirements.

Algorithmic considerations Each time step computed by ARC-3D may be divided into four distinct tasks: BC, RHS, Implicit, and Update, which are computed in that order. The boundary conditions are set in BC, the right-hand sides of the equations are evaluated in RHS, the implicit systems are set up and solved in Implicit, and the updates are performed in Update. The computations in RHS and Update are local in nature; at each grid point of the problem domain, the computations are performed using the information from the nearest neighbors defined by the 13-point stencil. The computations in BC require nearest-neighbor information as well as the solution of tridiagonal systems. Implicit consists of the solution of either scalar pentadiagonal systems or block tridiagonal systems. Over 96% of the computational work in ARC-3D is in Implicit and RHS, which is typical of similar CFD applications. Clearly, to achieve good performance, these two tasks must be parallelized well. On a sequential machine, BC and Update add an insignificant amount of work. However, BC and Update, when parallelized, add a significant communication overhead. Moreover, for unipartitions, BC causes a considerable amount of load imbalance. For these reasons, all four tasks must be parallelized so that the total overhead is minimized.

Implementation All experiments were carried out on a 16-processor Victor. **Table 2** summarizes some of the results for a grid of size 30 × 12 × 30, which was the largest problem size that could be solved on a single processor. (In order to calculate speedup, one must execute the problem on one processor.) The results shown here are for a case in which the implicit solver was based on the solution of a scalar pentadiagonal system. The entire application, including the grid-generation part, was parallelized. The fully pipelined Thomas algorithm was used to parallelize Implicit. All computations were performed using 64-bit arithmetic, and the FORTRAN code was parallelized using the communication primitives provided by the 3L parallel FORTRAN (a FORTRAN compiler for transputers developed by 3L Ltd. [30]).

Results In Table 2, the performance of the 2D unipartitions and 2D multipartitions is compared for 4- and 16-processor execution. (The 2D multipartitioning could not be used with a larger number of processors because the memory overhead increases with the number of processors for this type of partitioning. For more than 16 processors, the memory overhead exceeded the available memory.) On Victor, the 2D unipartitioning scheme clearly performed better than the 2D multipartitioning scheme. For a detailed discussion of the observed performance, see [28]. Another result, not shown in the table, is that the former scheme is readily scalable. At first these results seem counterintuitive, since the multipartitioning scheme

operates at a finer level of granularity and has negligible data-dependency-delay effects. However, a detailed analysis showed that for the 2D multipartitions, the increased number of partitions added considerable communication and load imbalance overhead, which outweighed the benefits of negligible data-dependency-delay effects.

This application was a rather sobering demonstration of the programming cost of parallelization: The original serial code contained 4200 lines in 34 subroutines, whereas the total line count for the parallel version was 20 000 lines in 101 subroutines. This count also includes communication routines tailored for the application. Currently we are in the process of implementing the 3D unipartitioning and 3D multipartitioning schemes. These are much harder to implement, since they involve mapping on a 2D grid of processors. However, analysis has shown that these two schemes have lower overheads and they should deliver superior performance; at the same time they should enable us to make efficient use of a larger number of processors.

• *Solution of linear systems using conjugate gradient method*

The conjugate gradient is a widely used method for solving linear systems of equations for which the coefficient matrix is symmetric and positive definite. Such systems arise in solving elliptic partial differential equations such as those generated by finite element methods. In our implementation, the solution of a Poisson equation on a 2D unit square was used as the model problem. The domain decomposition technique was used to subdivide the problem domain into connected but disjoint subdomains. Each subdomain was assigned to a processor, and the processor was responsible for performing the computations involving the nodal variables of that subdomain. With this type of partitioning, some information exchange is required between the processors containing neighboring partitions. For the model problem, the arithmetic work in one iteration of the conjugate gradient algorithm is comprised of one matrix-vector multiplication, three SAXPY operations, and two dot-product operations. These three component costs depend on the problem discretization and the implementation methods used. In [31], analytical expressions are derived for these components in terms of CPU speeds, message latency, data transmission speeds, and the diameter of the underlying interconnection network. On a 64-processor Victor, the largest problem that could be solved had 641 601 unknowns. This corresponded to the solution of a 2D Poisson equation on the unit square with 800 × 800 bilinear element discretization. The programs were written using the Express environment. For this problem, we found the efficiency with the 64-processor Victor to be 0.976, where the efficiency is defined as the speedup divided by the number of processors. In determining the efficiency, we had to extrapolate the single-processor computation time from the computation time of a smaller problem, since the model problem did not fit in the single-processor memory.

• *High-temperature superconductivity*

High-temperature superconductors are doped perovskite structure copper oxides, in which the key structure is the CuO plane. Electrons moving in that plane are believed to sustain a very strong short-range interelectronic repulsive interaction U. Because of the strength of U, there is no analytic procedure for solving the many-particle problem on a lattice. Thus, an attractive alternative is to invoke *ab initio* quantum Monte Carlo (QMC) methods. Simulations have been carried out on Victor by parallelizing a projection Monte Carlo method modeling lattices with 200 atoms for 100 time steps [32]. Two versions of the code were written: In the first version, each processor modeled all particles, and moves were carried out in parallel, whereas in the second version, each processor was associated with one particle. Calculations done with the first version demonstrated that a widely used phonon-free model of the CuO plane does not show any evidence of superconductivity. The major result obtained was that the projection Monte Carlo technique is feasible, overcoming earlier concerns that sign oscillations of the many-fermion system would render it inoperative. The second version of the algorithm (which is not subject to serious memory constraints), if run on much faster Victor processors than currently available, might provide a feasible path to an *ab initio* understanding of high-temperature superconductors. The algorithms show essentially linear speedup up to the 256 nodes of V256 and an absolute performance comparable to that of a Cray YMP. The code was written in FORTRAN.

• *Environmental modeling*

A domain-decomposition algorithm was applied for the parallel solution of the time-dependent shallow-water equations for wind-driven oceanic circulation. A conservation of pollutant-mass equation is also included in the shallow-water equation set, in order to study the transport of water-borne contaminants by the wind-generated oceanic currents.

A series of one-dimensional fast Fourier transforms (1D FFTs) in each dimension was used as part of the computational scheme for the solution of the equations, using Fourier or Chebyshev pseudospectral methods. These methods consist in expanding the unknown dependent variables in a global series of orthogonal and complete sets of functions (basis functions) and requiring that all of the differential equations be exactly satisfied at a set of points in the space domain (grid or collocation points). In most problems where gradients of the dependent variables must be evaluated, FFTs are

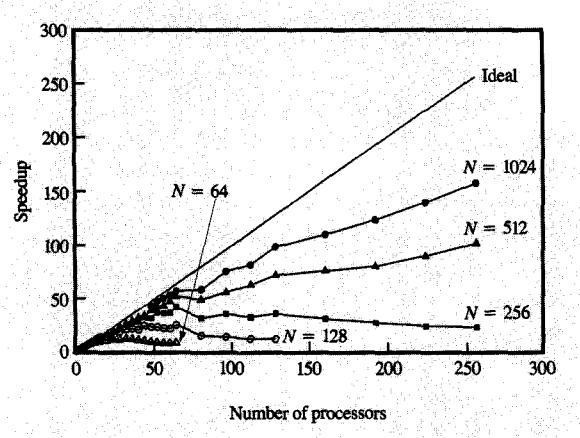

Figure 7

Speedup achieved by parallel solution of time-dependent shallow-water equations (N is grid dimension).

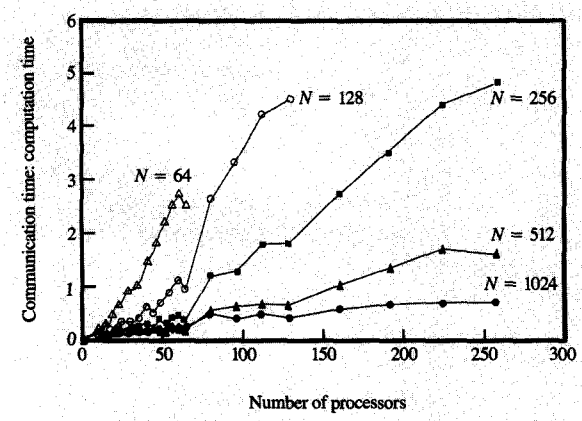

Figure 8

Ratio of communication time to computation time for shallow-water equation solution (N is grid dimension).

performed and local derivatives are evaluated analytically in the spectral domain. In turn, by the application of inverse FFTs, spectral gradients are transformed to grid-point gradients with a high order of accuracy. Parallelism is achieved not by a decomposition of the FFT algorithm but through the segmentation of the data as evenly as possible among processing elements (PEs), followed by the concurrent application of 1D FFTs on arrays resident in each PE. The need to evaluate gradients of the dependent variables in both directions requires the application of the 1D FFTs across the processors, demanding data communication among PEs. Once the gradients have been calculated, explicit time-marching numerical schemes can be applied independently in each processor partition without undue concern for the size of the grid. Because of the global communication demands of the application, and in order to achieve high communication bandwidths, efficient data-partitioning, data-blocking, and communication algorithms were implemented. These algorithms are portable (and able to be optimized conveniently for fast migration of sequential code with similar communication requirements) to distributed-memory systems.

Actual calculations were carried out on the Victor V256 multiprocessor. The code was written in FORTRAN using the Express CUBIX environment to communicate data-blocks among processors. Communication was achieved in each processor by sending and receiving appropriate messages to and from all participating processors. Express is responsible for routing the messages to the requested destinations. The routing process degrades the performance of the parallel code, since each message may be passed through several processors before it reaches its final destination. Furthermore, contention and latency also contribute to performance degradation in the parallel code.

Different test runs were performed, and speedup curves were obtained for square grids of dimension 64, 128, 256, 512, and 1024. As shown in **Figure 7**, the number of processors was varied from 1 to 256. A speedup of almost 160 was achieved on the 1024 × 1024 grid with 256 processors. This was due to the relatively large number of grid points per processor, which resulted in a large ratio of computation to communication (see **Figure 8**). Runs with smaller grid size (64 and 128) were performed with fewer processors since they could not be partitioned efficiently on a 256-node system. A slope reversal on the speedup curve was realized when the number of grid points per processor approached 1. This effect is due to message traffic, network contention, and latency. When the grid points could not be evenly partitioned among the processors, the computational load was unevenly balanced, resulting in the observed inflection points on the speedup curves (where N is not perfectly divisible by P).

- *Logic fault simulation*

Fault simulation is the simulation of a logic design that has been modified to reflect the presence of a fault, such as an open wire. Very many of these simulations must be performed to ensure detection of all plausible faults that could possibly occur during the manufacturing of a circuit. Fault simulation was parallelized for Victor by assigning

small clusters of gates to processors. Before a processor can execute the fault-simulation operations on a cluster G, it needs all relevant inputs from clusters logically preceding G; however, if two clusters are independent, the evaluation of both can proceed simultaneously. This parallelism has been exploited. Control of the fault-simulation process is distributed among the processors by structuring the simulation as a dataflow process. Upon completion of operations on a given cluster G, the processor responsible for G sends result messages to all processors requiring the results from G. This form of parallel fault simulation is very communication-intensive, and the overall performance is limited by overhead of the present store-and-forward message-passing scheme. Total speedups achieved were 35 to 40 on 256 nodes (but not showing signs of leveling off as the number of processors approached 256). In comparison to this, a simulation using a centralized-control-structure scheme "saturated" at a speedup of about 20 to 25, irrespective of the size of the design [33]. The code was written in Occam.

• *Parallel waveform relaxation*

Large-scale circuit simulation is an application that has outgrown available computational resources. Conventional tools permit designers to simulate circuits with up to about 5000 transistors, whereas circuits themselves often exceed several hundred thousand transistors. In the Parallel Waveform Relaxation project, large digital MOS circuits are algorithmically partitioned into smaller subcircuits that can be analyzed separately [34]. After all subcircuits have been analyzed, node voltage waveforms are shared among the subcircuits, as in the fault simulation, and a new analysis is initiated. When all waveforms show little change from one analysis to the next, the simulation is complete. The key issues in obtaining high parallel efficiencies for this problem are workload distribution and communication load. A static assignment of subcircuits to processors so that all processors have about the same number of transistors gives reasonably good workload balance. Communication is minimized by assigning chains of subcircuits that share many waveforms to the same processors. Another unusual feature of the code is the implementation of a mixed Gauss–Seidel/Gauss–Jacobi algorithm. The relaxation converges rapidly by using Gauss–Seidel initially but switches to Gauss–Jacobi once a processor runs out of subcircuits that can be solved with Gauss–Seidel. Currently, the simulator code runs on any number of processors and has been tested with a wide range of circuits, from 282 transistors in an arithmetic-logic-unit circuit to the logic portion of a 16MB DRAM design, with 186 000 transistors. Speedups of up to 200 have been achieved with 256 processors. The largest simulations run ten times faster than similar sequential code on a state-of-the-art mainframe. The code was written in C.

• *Processing strategies for large object-oriented databases*

The purpose of this project was to study the suitability of the message-passing paradigm for large, object-oriented databases. A simulator was implemented on Victor to test algorithms [35] for correctness and performance. The simulator executes router, controller, query, and one or more disk-emulator processes on each node. The host compiles user queries and initiates the actual processing on the appropriate node. The utilization of resources and the overall query execution times are recorded. The simulation results indicate that general-purpose message-passing systems can be effectively used for implementing large object-oriented databases. For the algorithms developed, it was observed that the retrieval of data from the secondary-storage devices was the factor limiting performance, even for very complex queries. Several slow disks per node were found to be preferable to one fast disk per node. The communication network limits the performance only for applications managing very complex objects or applications having high interconnectivity among data objects in the final stages of the query processing.

• *Multirobot simulation*

Multirobot simulation enables a user to view graphically the motion and interaction of multiple general robot arms in a simulated world. This is a problem of considerable practical interest, since clashes between real robot arms working simultaneously on a given task (e.g., populating a printed circuit board) are to be avoided at all costs. The problem was parallelized in a coarse-grain sense, with a path-planner process and a graphics process associated with each robot to be simulated. The outputs of these processes were directed to the general graphics server process, which drew the actual picture. For reasons of performance and ease of debugging, a small custom-made transputer operating-system kernel (TOPS) was written [36], and implemented in Parallel-C from Logical Systems. The kernel provides blocking and nonblocking message-passing, broadcasting, routing on a fixed 2D-mesh topology, fast dynamic-memory-space allocation, event synchronization, and various degrees of debugging and remote I/O support. The actual code for robot simulation and user control reside on top of this kernel. Measured performance of the system for the simulation of two robots, each with one prismatic and five revolving joints mounted on a spinning table, running on a Victor V16 system, was an animation rate of 2.8 pictures per second. Because of

Table 3 Application speedups on Victor.

Applications	*Number of processors*	*Observed speedups*
3D CFD application	16	13
Conjugate gradient	64	62
Environmental modeling	256	170
High-temperature superconductivity	256	≈256
Logic fault simulation	256	40
VLSI waveform relaxation	256	200

the very coarse-grained nature of the problem decomposition, one would not expect higher performance with more nodes.

• *Summary*

In **Table 3**, we summarize the performance in terms of speedups of some of the applications described above. The highest performance measured on the V256 system was 224 MFLOPS (32-bit arithmetic) with an application (fractals) written in Occam. More typical applications using FORTRAN or C compilers and 64-bit arithmetic, with speedups over 150, achieved about 50 MFLOPS. Many of these applications cannot be organized to perform well on a vector supercomputer. From our experience, these applications parallelized well, even on a first-generation distributed-memory machine such as V256. In some cases, the performance on V256 was superior to that obtained on a mainframe or a vector supercomputer. It has been the objective of this project, however, to provide a test bed for gaining experience in efficiently parallelizing real applications on a machine with hundreds of nodes. Toward that end, we have succeeded very well in achieving our objectives. In these experiments, several new algorithms and parallelization methods were implemented and their performance analyzed. It was also encouraging to see that very high speedups (over 200) were achieved for important applications.

4. Conclusions

Here we present some evolving insights stimulated by our work on the Victor project. Many of these findings are being applied to Vulcan, the next generation of message-passing multiprocessors currently being developed in the Parallel Systems Department at the IBM Thomas J. Watson Research Center.

• *Parallelism and performance*

Our most important observation is that many large scientific and engineering problems exhibit abundant intrinsic parallelism. While it is true that applications for parallel machines are often selected just because they are easy to parallelize, several of the applications studied on Victor—particularly the ones related to electronic design automation—were chosen rather because they represent large CPU workloads within IBM. It was encouraging to see that these real problems could be parallelized quite readily.

The parallelism in large applications appears on many different levels: from very coarse-grained task parallelism to fine-grained parallelism found within tight loops. Human experts familiar with an application usually find it very easy to identify coarse-grained parallelism, whereas compilers are making good progress in finding fine-grained and loop-level parallelism. Thus, humans and compilers complement each other in finding and exploiting parallelism, but much work must still be done to develop better methods for expressing human knowledge about parallelism to the computer.

Another recurring theme in all of the applications studied—which furthers a point first stated clearly by Gustafson et al. [37]—is that it is possible to increase the number of processors with the problem size and retain constant parallel efficiency over a wide range of processor numbers. This observation holds true if the workload per processor remains constant and neither communication nor synchronization overhead is the factor limiting performance.

This insight also adds substance to the widely held belief that a practical market niche for parallel machines exists at the very high end, where the difficulties of writing parallel code are accepted by users because there are no alternatives. There is no doubt now that several of the applications currently running on or being developed for V256 will scale gracefully to several thousand processors, on machines with a similar ratio of communication capabilities to computation capabilities. We do not yet know, however, whether this scaling will extend well to tens of thousands of nodes.

Discussion with scientific users of the Victor machines has made it clear that they do not mind developing a parallel program "from scratch" if there is a payback in the form of *at least* a tenfold performance improvement over existing machines. At present, this is not the case; the measured performance of V256 and other large parallel machines [38] is still only comparable to that of existing supercomputers. (For a succinct and thoughtful definition of what a *supercomputer* is, see the paper by Hey [11].) In the user community there is considerable faith, however, that the next generation of parallel machines will deliver the long-hoped-for performance gains promised by parallel computing. Thus, users are willing to develop complex parallel code today if they believe that the code will be portable to future systems. As a result, there has been very high interest among Victor users in the portable Express environment [24], even though there is currently a considerable communication performance penalty in using it [39].

We have found four significant reasons why speedups may not be linear in the number of processors:

- Insufficient problem parallelism.
- Communication bottlenecks.
- Synchronization overhead.
- Workload imbalance.

The first two issues have already been discussed. For the applications running on Victor, the loss in efficiency due to synchronization overhead was generally found to be below 10%. A contributing factor to permitting this surprisingly small loss is the fact that most large applications have so much *excess parallelism* that processors usually manage to stay busy even though several individual processes may be blocked.

The problem of balancing workload has proven to be less troublesome than anticipated. For regular problems, it is generally straightforward to devise a work-assignment plan that prevents workload imbalance. For irregular problems, the simple dynamic balancing scheme described in Section 3 in the subsection on fractals and ray tracing worked in several cases. Simple algorithms were found to work satisfactorily for the remaining applications. More sophisticated techniques, such as dynamic task migration, have not proven to be necessary.

- *Victor system capabilities*

There must be sufficient memory per node—not less than what is found on state-of-the-art workstations. This has been the most painful and one of the most telling lessons of the Victor project. Any MIMD computer that seeks a significant user community must be able to run large, unmodified serial code on each node without users being limited by memory-space restrictions. This implies virtual-memory capability on all nodes, which, in turn, requires a large backing-store capability.

With regard to the time required to pass messages from processor to processor, there is no consensus among users whether all processors on the network should appear equidistant (as in a multistage interconnect network) or not (as for a mesh or hypercube). Users with irregular problems most often prefer the former topology, while those with regular problems favor the latter.

The visual hardware monitor system has been extremely valuable in developing and debugging parallel programs. This is surprising, since the monitor does not recognize software events such as context switching, but only hardware conditions such as a communication link being busy. Two bits per node are displayed by the monitor, allowing some visual indication of program state. The global system view provided by the monitor aided in many complex software-debugging situations. The fact that the operation of the monitor cannot be disturbed by any user action has also been vital.

The transputer provides asynchronous communication hardware, and there is no system-wide clock in Victor. Our experience with the reliability of such a potentially metastable system has been extremely good. It appears that it will be possible to use such asynchronous systems for a very large number of processors. Except for the inconvenience of not being able to generate precise global timestamps for event tracing, the lack of a global clock has not been a problem in using the machine.

Another noteworthy feature of the Victor architecture is the system partitioning, which permits multiuser support. It has been the experience of our group that a machine of the size of V256 must be a shared resource. There is no need to support a large number of users, but a system in which a few users divide the resources of the machine in a spatial—not a temporal—sense appears to hold promise for the high-performance market. The overhead required for protection in timesharing systems affects context-switching and message-latency times, both of which are very sensitive parameters in a parallel computer. But allowing individual users to use different parts of the system, with only simple mechanisms to isolate them from one another, has been very effective.

The lack of high-performance disk I/O and the limited memory per node on most parallel systems may be reasons why there are very few parallel production programs available yet. Fortunately, many scientific and engineering applications are characterized by sequential accesses to large files, and it is relatively straightforward to parallelize disk I/O for access to such files. On Victor, the parallel waveform relaxation application employs the file system. It uses the 16 disk nodes in a straightforward, independent manner. The 16 processing nodes in a column utilize the disk node in that column as the working data input and output device. All program files are still kept on the host node disks. The entire field of parallel file I/O and virtual-memory support is an active research area and will be so for a long time to come.

- *Final statement*

In summary, we found that a carefully designed message-passing multiprocessor system can productively run a wide range of programs. We found that the software effort required to run code on such a machine was reasonable and worth undertaking in view of potential performance gains. One great challenge of the early nineties will be to realize the promise of large-scale parallelism to solve compelling scientific and engineering problems.

Acknowledgments

The authors thank their colleagues and visitors at the IBM Thomas J. Watson Research Center for many helpful

discussions during application development. In particular we are indebted to Genevieve Cerf, Hubertus Franke, Steven Glim, Grant Haab, Leendert M. Huisman, Thomas A. Johnson, Ramesh Natarajan, Pratap Pattnaik, and Arun K. Thakore, who developed many of the applications discussed and who contributed to Section 3. We thank Eva Ma for her many contributions to the development of E-kernel. We also thank our guests from other IBM locations and the university professors and students who have worked on Victor, but who are far too many to name individually. We thank the Central Scientific Services Department at the IBM Thomas J. Watson Research Center, especially William J. Goss, Robert V. Lemay, Jean A. Mar, and Thomas Picunko, for their tireless efforts during system fabrication. Finally, we gratefully acknowledge the continuing management support from Zeev Barzilai at the IBM Thomas J. Watson Research Center.

References

1. Richard E. Matick, "Memory and Storage," *Introduction to Computer Architecture*, Harold S. Stone, Ed., Science Research Associates, Chicago, 1980, Ch. 5.
2. *The Transputer Databook*, Inmos Corporation, Bristol, UK, 1989.
3. W. W. Wilcke, D. G. Shea, R. C. Booth, D. H. Brown, M. E. Giampapa, L. Huisman, G. R. Irwin, E. Ma, T. T. Murakami, F. T. Tong, P. R. Varker, and D. Zukowski, "The IBM Victor Multiprocessor Project," *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, Monterey, CA, March 1989, pp. 201–207.
4. Charles L. Seitz, "The Cosmic Computer," *Commun. ACM* **28,** No. 1, 22–33 (January 1985).
5. Mark Homewood, David May, David Shepherd, and Roger Shepherd, "The IMS T800 Transputer," *IEEE Micro* **7,** No. 5, 10–26 (October 1987).
6. Shekhar Borkar, Robert Cohn, George Cox, Thomas Gross, H. T. Kung, Monica Lam, Margie Levine, Brian Moore, Wire Moore, Craig Peterson, Jim Susman, Jim Sutton, John Urbanski, and Jon Webb, "Supporting Systolic and Memory Communication in iWarp," *Proceedings of the 17th Annual International Symposium on Computer Architecture*, IEEE Computer Society Press, May 1990, pp. 70–81.
7. William J. Dally, Andrew Chien, Stuart Fiske, Waldemar Horwat, John Keen, Michael Larivee, Rich Lethin, Peter Nuth, Scott Wills, Paul Carrick, and Greg Fyler, "The J-Machine: A Fine-Grain Concurrent Computer," *Information Processing 89*, G. S. Ritter, Ed., Elsevier Science Publishers North-Holland, 1989, pp. 1147–1153.
8. James T. Rayfield, *Armstrong, a Loosely-Coupled Multiprocessor Testbed for Reconfigurable Topologies*, Ph.D. thesis, Brown University, Providence, RI, June 1988.
9. M. Aspnas, R. J. R. Back, and T. E. Malen, "The Hathi-2 Multiprocessor System," *Technical Report Series A*, No. 80, Abo Akademi, Oulu, Finland, June 1989.
10. *Transputer Applications*, Gordon Harp, Ed., Pitman, London, 1989.
11. Anthony J. G. Hey, "Supercomputing with Transputers—Past, Present and Future," *Proceedings of the 1990 International Conference on Supercomputing*, Amsterdam, ACM Press, September 1990, pp. 479–489.
12. C. Clos, "A Study of Nonblocking Switching Networks," *Bell Syst. Tech. J.* **32,** 406–424 (1953).
13. G. Langdon, Jr., *Computer Design*, IEEE Computer Press, San Jose, CA, 1982.
14. Eva Ma and Ramesh Krishnamurti, "The Architecture of REPLICA—A Special-Purpose Computer System for Active Multi-Sensory Perception of 3-Dimensional Objects," *Proceedings of the 11th Annual International Symposium on Computer Architecture*, Ann Arbor, MI, 1984, pp. 30–37.
15. Geraint Jones and Michael Goldsmith, *Programming in Occam 2*, Prentice-Hall International, New York, 1988.
16. David May and R. Taylor, "Occam—An Overview," *Microproc. & Microsyst.* **8,** No. 2, 73–79 (March 1984).
17. C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall International, Bristol, UK, 1985.
18. C. A. R. Hoare, "Communicating Sequential Processes," *Commun. ACM* **21,** No. 8, 666–677 (August 1978).
19. Eva Ma and Dennis G. Shea, "The Embedding Kernel on the IBM Victor Multiprocessor for Program Mapping and Network Reconfiguration," *Proceedings of the 2nd IEEE Symposium on Parallel and Distributed Processing*, December 1990, pp. 874–879.
20. Eva Ma and Dennis G. Shea, "E-Kernel—An Embedding Kernel on the IBM Victor Multiprocessor for Program Mapping and Network Reconfiguration," *Research Report RC-16771*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, April 1991.
21. Dennis G. Shea, "E-Kernel on the IBM Victor V256 Multiprocessor—An Experimental Platform for Parallel Systems," Ph.D. thesis, University of Pennsylvania, Philadelphia, December 1991.
22. Lixin Tao and Eva Ma, "Simulating Parallel Neighboring Communications Among Square Meshes and Square Toruses," *J. Supercomputing* **5,** No. 1, 57–71 (1991).
23. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Computers*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1988.
24. *Express User's Manual*, Parasoft Co., 2500 E. Foothill Blvd., Pasadena, CA 91107, 1989.
25. Winfried W. Wilcke, "A General Purpose Code for Monte Carlo Simulations," *Nuclear Instrum. & Meth. Phys. A* **225,** 132–137 (August 1984).
26. G. Cerf, D. G. Shea, W. W. Wilcke, and D. J. Zukowski, "Parallelizing Back-Propagation on IBM's Victor V32 Multiprocessor," *Research Report RC-13788*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, April 1988.
27. T. H. Pulliam and J. L. Steger, "Recent Improvements in Efficiency, Accuracy, and Convergence for Implicit Approximate Factorization Algorithms," presented at the *AIAA 23rd Aerospace Science Meeting*, Reno, NV, 1985.
28. V. K. Naik, N. H. Decker, and M. Nicoules, "Implementation of Implicit Scheme Based CFD Applications on Message Passing Systems," *Parallel CFD: Implementations and Results*, H. Simon, Ed., MIT Press, Cambridge, MA, 1991. (Also available as *Research Report RC-16567*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1992.)
29. V. K. Naik, "Performance Effects of Load Imbalance in Parallel CFD Applications," *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing*, D. Sorenson, Ed., Philadelphia, 1991.
30. *3L Parallel FORTRAN User Guide*, 3L Ltd., Peel House, Ladywell, Livingstone EH54 6AG, Scotland, 1988.
31. Ramesh Natarajan and Pratap Pattnaik, "Performance of the Conjugate Gradient Method on Victor," *Research*

Report RC-16328, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, November 1990.
32. M. Frick, P. C. Pattnaik, I. Morgenstern, D. M. Newns, and W. von der Linden, "Monte Carlo Study of Superconductivity in the Three-Band Emery Model," *Phys. Rev. Lett.* **42,** No. 4, 2665–2668 (August 1990).
33. Leendert Huisman, Indira Nair, and Raja Daoud, "Fault Simulation of Logic Designs on Parallel Processors with Distributed Memory," *Proceedings of the International Test Conference*, Washington, DC, September 1990, pp. 690–697.
34. T. A. Johnson and D. J. Zukowski, "Waveform-Relaxation-Based Circuit Simulation on the Victor V256 Parallel Processor," *IBM J. Res. Develop.* **35,** No. 5/6, 707–720 (1991, this issue).
35. A. K. Thakore, S. Y. W. Su, H. Lam, and D. G. Shea, "Asynchronous Parallel Processing of Object Bases Using Multiple Wavefronts," *Proceedings of the 1990 International Conference on Parallel Processing*, Vol. 1, Penn State Press, August 1990, pp. 127–135.
36. H. Franke, D. G. Shea, and L. C. Zai, "Transputer-Based Multi Robot Simulation," *Proceedings of the Third Conference of the North American Transputer Users Group*, Alan S. Wagner, Ed., IOS Press, Sunnyvale, CA, April 1990, pp. 139–149.
37. John L. Gustafson, Gary R. Montry, and Robert E. Benner, "Development of Parallel Methods for a 1024-Processor Hypercube," *SIAM J. Scientif. & Statist. Computing* **9,** No. 4, 609–638 (July 1988).
38. P. Messina, C. Ballie, P. Hipes, R. Williams, A. Alagar, A. Kamrath, R. Leary, W. Pfeiffer, J. Rodgers, and D. Walker, "Benchmarking Advanced Architecture Computers," *Concurrency: Pract. & Exper.* **2,** No. 3, 195–255 (1990).
39. L. R. Hu and G. S. Stiles, "Fluid Dynamics on Express: An Evaluation of a Topology-Independent Parallel Programming Environment," *Transputer Research and Applications*, D. L. Fielding, Ed., 1990, pp. 1–8.

Received February 18, 1991; accepted for publication August 28, 1991

Dennis G. Shea *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Shea is a research staff member and manager of the Modular Microsystems group at the IBM Thomas J. Watson Research Center, where the Victor V256 system was developed. He joined IBM Boca Raton in 1979, with a B.S.E.E. and M.E.E.E. from Rensselaer Polytechnic Institute. While at Boca Raton, he worked in the areas of LSI component engineering and new systems advanced technology and completed an M.A.S. degree in computer systems at Florida Atlantic University. He left Boca via an IBM resident study fellowship and received his Ph.D. from the Department of Computer and Information Science at the University of Pennsylvania. Dr. Shea is a member of Tau Beta Pi and Eta Kappa Nu.

Winfried W. Wilcke *HaL Computer Systems, 1315 Dell Avenue, Campbell, California 95008.* Dr. Wilcke was senior manager at the IBM Thomas J. Watson Research Center in charge of the Microsystems Architecture area. He holds a Diplom and a Ph.D. in nuclear physics (J. W. Goethe University, 1976) and spent several years on the faculty of the University of Rochester. His research, which has resulted in over 180 publications, centered on experimental studies of nuclear reactions at the Lawrence Berkeley Laboratory and the Los Alamos Scientific Laboratory. In 1984, Dr. Wilcke's interest turned to microelectronics and computer architecture; he joined the IBM Thomas J. Watson Research Center, where he worked on VLSI microsystems and the design of large-scale multiprocessors for scientific applications. He is now Director of Architecture at HaL Computer.

Richard C. Booth *IBM Application Business Systems, 3605 Highway 52 North, Rochester, Minnesota 55901.* Mr. Booth is a senior engineer with the processor architecture group at the ABS Development Laboratory in Rochester, Minnesota. He received his B.S. in electrical engineering from the University of Minnesota in 1974. Mr. Booth held technical leadership and management positions in the development of IBM System/34, IBM System/36, and IBM AS/400®. From 1986 until 1987, he completed a technical sabbatical at the IBM Thomas J. Watson Research Center, working on message-passing multiprocessor systems. Mr. Booth has received the IBM Outstanding Technical Achievement, Excellence, and Invention Achievement Awards. His continuing interests are in the areas of parallelism and commercial systems.

Dana H. Brown *IBM Application Business Systems, 3605 Highway 52 North, Rochester, Minnesota 55901.* Mr. Brown is an advisory engineer in the Compact Storage Laboratory in Rochester, Minnesota. He holds a B.S.E.E. degree from Pennsylvania State University and an M.S.E.E. degree from Northeastern University. Mr. Brown began his career working for Honeywell Information Systems in product memory design. He joined Boeing Aerospace to develop a large airborne computer and display system used on the AWACS aircraft. In 1977, he began work on low-end disk drives at IBM. Mr. Brown currently designs small-form-factor prototype disk drives employing advanced technology components. He is a member of IEEE.

Zaphiris D. Christidis *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Christidis is a research staff member at the IBM Thomas J. Watson Research Center. He received a B.S. in mathematics from the University of Ioannina, Greece, an M.A. in meteorology from City College of New York in 1980, and a Ph.D. in atmospheric science from the University of Michigan in 1986. In 1987 he joined IBM Kingston, where he worked toward the development of parallel algorithms for applications related to fluid dynamics. In 1988 he joined the IBM Thomas J. Watson Research Center; he is currently working on parallel applications related to air/water pollution and atmospheric/oceanic circulation modeling for distributed-memory parallel systems.

Mark E. Giampapa *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Giampapa is a member of the Parallel System Organization group at the IBM Thomas J. Watson Research Center. After graduating from Columbia University with a Bachelor's degree in computer science in 1984, he joined IBM Research to work on cooperative and distributed processing in the Advanced Workstations group. In 1987, he

joined the Parallel Microsystems Department to develop an architectural simulator for advanced RISC microprocessors and later joined the Victor project, focusing on system monitoring. His present research includes the design and implementation of distributed operating environments for massively parallel message-passing MIMD machines.

Gail Blumenfeld Irwin *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Ms. Irwin is a member of the Parallel System Organization group at the IBM Thomas J. Watson Research Center. She received her B.A. in mathematics from the Courant Institute at New York University and attended the Columbia University Graduate School of Electrical Engineering. Prior work experience at the IBM Thomas J. Watson Research Center includes AI/expert systems in the Mathematical Science Department. Ms. Irwin has also been involved in the development of TSO, PL/I compilers, and various other IBM products, compilers, and access methods at IBM locations in New York City, White Plains, and Boca Raton.

Thomas T. Murakami *IBM Entry Systems Technology Division, 1000 NW 51st Street, Boca Raton, Florida 33432.* Mr. Murakami was a member of the Modular Microsystems group at the IBM Thomas J. Watson Research Center, on assignment from the IBM Boca Raton facility. He joined IBM at Boca Raton in 1979 with a B.S. from Rutgers College of Engineering (1977) and an M.E. in electrical engineering from Rensselaer Polytechnic Institute (1979). At Boca Raton Mr. Murakami worked in the areas of LSI component engineering, circuit card engineering analysis, and card test engineering. He is a member of Tau Beta Pi and Eta Kappa Nu.

Vijay K. Naik *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Naik is a research staff member in the Parallel Applications group at the IBM Thomas J. Watson Research Center. Dr. Naik received A.M. and Ph.D. degrees in computer science from Duke University in 1984 and 1988, respectively. In 1980, he received a B.Tech. from the Indian Institute of Technology, Madras, and an M.S. from the University of Miami in 1982, both in mechanical engineering. Prior to joining IBM, he was a staff scientist at ICASE, NASA Langley Research Center. His current research interests include characterization of scientific applications and algorithms for scalable high-performance multiprocessor systems. In particular, he is interested in applications in the area of computational fluid dynamics and algorithms for sparse-matrix computations. He is a member of IEEE and Sigma Xi.

Fred T. Tong *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Tong received his B.S.E.E. from City College of New York in 1973 and his M.S.E.E. from Syracuse University in 1977. He joined IBM Endicott in 1973, and in 1981 became a research staff member in the Computer Science Department at the IBM Thomas J. Watson Research Center, where he worked on various System/370™- and RISC-based machine designs. He later worked on the Victor parallel processor project. Mr. Tong is currently a project leader of a high-performance VLIW prototype under development at the IBM Thomas J. Watson Research Center. He holds several patents related to virtual memory and cache subsystems. Mr. Tong is a member of Eta Kappa Nu and IEEE.

Philip R. Varker *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Varker is an electrical engineer with the Modular Microsystems group at the IBM Thomas J. Watson Research Center. He graduated with a B.S.E.E. from Cornell University in 1983. Since joining IBM Research, he has worked on a number of projects related to parallel processing, including the Yorktown Simulation Engine and the disk subsystem of Victor V256. His current research interests are high-performance I/O subsystems and hardware-monitoring techniques for parallel systems. Mr. Varker is a member of Tau Beta Pi and Eta Kappa Nu.

Deborra J. Zukowski *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Ms. Zukowski is a member of the Modular Microsystems group at the IBM Thomas J. Watson Research Center. After receiving her S.B. degree in electrical engineering from MIT in 1983, she worked at DEC's Eastern Research Laboratory as an internal consultant for performance issues of serial machines. In 1985, she joined the IBM Research Division, where she continued work with uniprocessor performance techniques. Ms. Zukowski has been investigating parallel circuit simulation since January 1989. Her interests include design and implementation of highly parallel architectures and applications, and performance issues of both parallel and serial processors.